Amendments to the Specification:

I. Amendments to Insert Section Headings

Please insert the following section heading at page 1, of the application, following the Cross Reference to Related Applications section added in the Preliminary Amendment filed on October 2, 2006, and prior to the first paragraph on page 1 of the originally filed application, beginning "The present invention relates to an operating system…" :

Technological Field

Please insert the following section heading at page 1, line 5 of the application between the first and second paragraphs on page 1:

Background

Please insert the following section heading following the first paragraph on page 6, ending with, "…directly into unreliability in the operating system as a whole," and prior to the second paragraph on page 6, beginning with, "It is therefore an object of the present invention to provide…":

Brief Summary

Please insert the following section heading on page 7 of the specification following the Description of the Drawings section added by way of the preliminary amendment filed on October 2, 2006, and prior to the second paragraph on page 7 of the originally filed specification, beginning with "An embodiment of the present invention will now be described…":

Detailed Description

Applicants note with respect to the above addition to the specification, that the preliminary amendment filed on October 2, 2006 refers to a different page and line numbering than referenced in the instant amendment. In this regard, the reference to page 8, line 4 in the preliminary amendment filed on October 2, 2006 corresponds to page 7, line 9 of the originally filed specification referred to in the instant amendment.

II. Amendments Related to Drawing Descriptions

A. Please amend the first paragraph on page 3 of the originally filed specification as follows:

Figure 1 illustrates the key difference between the two architectural paradigms described above. In this regard, Figure 1 illustrates an example layered monolithic kernel design 102 and an example microkernel design 104, in which the key difference between the two architectural paradigms may be seen.

B. Please amend the last paragraph beginning on page 7 of the originally filed specification as follows:

It is not considered necessary to fully describe the Symbian OS operating system in order to provide a sufficient understanding of this invention. Thus, the following description is restricted to those parts of the operating system relevant to the invention. The Symbian OS kernel is a hybrid between the monolithic 102 and microkernel 104 approaches shown in FIG. 1, and therefore combines certain advantages of both. The concept of a kernel which implements a message-passing framework for the benefit of user-side servers is derived from micro-kernel architectures; its networking and telephony stacks, as well as the file system, are all user-side servers. On the other hand, the implementation of device drivers as loadable kernel modules derives from monolithic kernel architectures.

C. Please amend the section on page 12, including the addition by the preliminary amendment of October 2, 2006 (note that the preliminary amendment referenced page 13, line 19, whereas the appropriate page and line reference is page 12, line 8, following the pseudocode for the algorithm for waiting on a fast mutex and prior to the paragraph beginning "In the case where there is no contention...") as follows:

Lock the kernel
Reenable interrupts
Current thread -> iWaitFastMutex = NULL
ENDIF
Current thread -> iHeldFastMutex = this
iHoldingThread = Current thread
Unlock the kernel

This procedure is shown in Figure 2. In this regard, as illustrated in Figure 2, operation 200 may comprise locking the kernel. Operation 202 may comprise checking the fast mutex's pointer to its holding thread. Operation 204 may comprise determining whether the pointer is clear (e.g., whether the fast mutex is free). If the pointer is clear, the procedure may skip to operation 216 (described below). If, however, the pointer is not clear, the procedure may proceed to operation 206. Operation 206 may comprise setting the fast mutex's contention/waiting flag. Operation 208 may comprise setting the current thread's mutex waiting pointer to this fast mutex. Operation 210 may comprise yielding to thread holding the fast mutex. On return from the holding thread to the current thread, the kernel is unlocked and interrupts are disabled. Operation 212 may comprise locking the kernel and re-enabling interrupts. Operation 214 may comprise clearing the current thread's mutex waiting pointer. Operation 216 may comprise setting the current thread's held mutex pointer to this mutex. Operation 218 may comprise setting the fast mutex's pointer to its holding thread to the current thread. Operation 220 may comprise unlocking the kernel. Operation 222 may comprise the end of the

procedure.

D. Please add the following paragraph following the first full paragraph on page 12 of the specification, which concludes with "...has been further optimized by simply disabling interrupts rather than locking the kernel while checking iHoldingThread. This procedure is shown in Figure 4.":

In this regard, as illustrated in Figure 4, operation 400 may comprise locking disabling interrupts. Operation 402 may comprise checking the fast mutex's pointer to its holding thread. Operation 404 may comprise determining whether the pointer is clear (e.g., whether the fast mutex is free). If the pointer is clear, the procedure may skip to operation 414 (described below). If, however, the pointer is not clear, the procedure may proceed to operation 406. Operation 406 may comprise setting the fast mutex's contention/waiting flag. Operation 408 may comprise setting the current thread's mutex waiting pointer to this fast mutex. Operation 410 may comprise yielding to thread holding the fast mutex. Operation 412 may comprise clearing the current thread's mutex waiting pointer. Operation 414 may comprise setting the current thread's held mutex pointer to this mutex. Operation 416 may comprise setting the fast mutex's pointer to its holding thread to the current thread. Operation 418 may comprise re-enabling interrupts. Operation 420 may comprise the end of the procedure.

E. Please amend the section on page 13, including the addition by the preliminary amendment of October 2, 2006 (note that the preliminary amendment referenced page 15, line 14, whereas the appropriate page and line reference is page 13, line 26, following the pseudocode for the algorithm for releasing a fast mutex and prior to the paragraph beginning "In the case where there is no contention...") as follows:

Lock the kernel
iHoldingThread = NULL
Current thread -> iHeldFastMutex = NULL

```
IF iWaiting
        iWaiting = FALSE
        Set TheScheduler.iRescheduleNeededFlag to cause reschedule
        IF CurrentThread->iCsFunction && CurrentThread->iCsCount==0
                Do critical section exit processing for current thread
        ENDIF
ENDIF
Unlock the kernel
```

This procedure is shown in Figure 3. In this regard, as illustrated in Figure 3, operation 300 may comprise locking the kernel. Operation 302 may comprise clearing the fast mutex's pointer to its holding thread. Operation 304 may comprise clearing the fast mutex's pointer to its holding thread. Operation 306 may comprise determining whether the fast mutex's contention/waiting flag is set. If the flag is not set, the procedure may skip to operation 318 (described below). If, however, the flag is set, the procedure may proceed to operation 308. Operation 308 may comprise clearing the fast mutex's contention/waiting flag. Operation 310 may comprise setting the appropriate scheduler flag to cause a reschedule. Operation 312 may comprise determining whether the thread is in a critical section. If the thread is not in a critical section, the procedure may skip to operation 318 (described below). If, however, the thread is in a critical section, the procedure may proceed to operation 314. Operation 314 may comprise determining whether the thread has a delayed function to be performed. If the thread does not have a delayed function to be performed, the procedure may skip to operation 318 (described below). If, however, the thread does have a delayed function to be performed, the procedure may proceed to operation 316. Operation 316 may comprise performing critical section exit processing for the current thread. Operation 318 may comprise unlocking the kernel. Operation 320 may comprise the end of the procedure.

E. Please add the following paragraph on page 14 of the originally filed specification, following the paragraph concluding with "...hence the check of iCsCount

and iCsFunction in the mutex release code," and prior to the paragraph beginning with, "The nanokernel includes a scheduler...":

Referring again to Figure 4, operation 450 may comprise disabling interrupts. Operation 452 may comprise clearing the fast mutex's pointer to its holding thread. Operation 454 may comprise clearing the fast mutex's pointer to its holding thread. Operation 456 may comprise determining whether the fast mutex's contention/waiting flag is set. If the flag is not set, the procedure may skip to operation 468 (described below). If, however, the flag is set, the procedure may proceed to operation 458. Operation 458 may comprise clearing the fast mutex's contention/waiting flag. Operation 460 may comprise setting the appropriate scheduler flag to cause a reschedule. Operation 462 may comprise determining whether the thread is in a critical section. If the thread is not in a critical section, the procedure may skip to operation 468 (described below). If, however, the thread is in a critical section, the procedure may proceed to operation 464. Operation 464 may comprise determining whether the thread has a delayed function to be performed. If the thread does not have a delayed function to be performed, the procedure may skip to operation 468 (described below). If, however, the thread does have a delayed function to be performed, the procedure may proceed to operation 466. Operation 466 may comprise performing critical section exit processing for the current thread. Operation 468 may comprise re-enabling interrupts. Operation 470 may comprise the end of the procedure.

III. Amendment to the Title

Please amend the title as follows:

~~IMPROVEMENTS IN OR RELATING TO~~ AN OPERATING SYSTEM PROVIDING A MUTUAL EXCLUSION MECHANISM ~~FOR A COMPUTING DEVICE~~